UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/676,373	09/30/2003	Stefan Jesse	09700.0216-00	3224

60668 7590 12/13/2011
SAP / FINNEGAN, HENDERSON LLP
901 NEW YORK AVENUE, NW
WASHINGTON, DC 20001-4413

EXAMINER	
VU, TUAN A	
ART UNIT	PAPER NUMBER
2193	

MAIL DATE	DELIVERY MODE
12/13/2011	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

# Office Action Summary	Application No.	Applicant(s)	
	10/676,373	JESSE ET AL.	
	Examiner	Art Unit	
	TUAN VU	2193	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE _3_ MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on _03 October 2011_.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ An election was made by the applicant in response to a restriction requirement set forth during the interview on _____; the restriction requirement and election have been incorporated into this action.

4)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

5)☒ Claim(s) _1,3,4,6-10,12,14-18 and 20-26_ is/are pending in the application.

5a) Of the above claim(s) _____ is/are withdrawn from consideration.

6)☐ Claim(s) _____ is/are allowed.

7)☒ Claim(s) _1, 3-4, 6-10, 12, 14-18, 20-26_ is/are rejected.

8)☐ Claim(s) _____ is/are objected to.

9)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

10)☐ The specification is objected to by the Examiner.

11)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

12)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

13)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)	4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)	5) ☐ Notice of Informal Patent Application
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.	6) ☐ Other: _____.

DETAILED ACTION

1. This action is responsive to the Applicant's response filed 10/03/11.

As indicated in Applicant's response, claims 1, 3-4, 10, 18, 20-23 have been amended. Claims 1, 3-4, 6-10, 12, 14-18, 20-26 are pending in the office action.

Claim Rejections - 35 USC § 103

2. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

3. Claims 1, 3-4, 6-7, 10, 12, 14-18, 23-26 are rejected under 35 U.S.C. 103(a) as being unpatentable over Mestre et al, USPubN: 2004/0015834 (herein Mestre) in view of Fry et al, USPubN: 2003/0163603 (herein Fry), further in view of Worden, USPubN: 2003/0149934 (herein Worden) and Severin, USPubN: 2005/0005261 (herein Severin).

As per claim 1, Mestre discloses a computer-readable storage device storing a computer program product operable to cause a data processing apparatus to:

receive a metamodel in a first language (UML – para 0080, pg. 4), the metamodel describing a diagram of classes that define one or more development objects (para 0090, pg. 4), the development objects representing building blocks for developing an application (Note: Rational Rose in UML – see Modeling objects, Rose 8, model.mdl - Fig. 3 -- reads on building blocks for development of an application, see: Application 50 use - Fig. 3; *software development process, business process of interest* - para 0005, pg. 1; para 0033);

convert the metamodel to a model description in a second language (XML - para 0078, pg. 3) according to an interchange format (XMI – para 0080, pg. 4; para 0086 pg. 4)

generate a set of intermediate objects to represent the classes of the metamodel (*classes*, associated objects, attribute – para 0089-0092; XIDL, bindings 18, 20, 22 - para 0088; self-contained list of classes – Fig. 2); and

generate code using the set of intermediate objects as inputs to derive an Application Program Interface (API), the API (para 0095-0102, pg. 5; helper, getters, setters, object to be retrieved, marhals, unmarshals – para 0102, pg. 5; Xbean, Xkey, XJava2XML, AXML2Java, ARDBHelper - Fig. 3; Generators Fig. 4) enabling access of development objects (e.g. Java code … allow use of those objects – para 0078, pg. 4; para 0096-0099; search, update, create, delete … transaction – para 100-0101 – Note: open architecture including integrating modeling tools, type conversion, reuse of messages, optimization of code, retargeting optimizes database queries – see para 0105-0106, pg. 5 – reads on java API generated to enable developments objects to be accessed or developed).

Mestre does not explicitly disclose generating a set of intermediate representing classes of the metamodel *by parsing the model description*. The UML model in Mestre's being transformed into a intermediate representation (para 0089-0093) entails association between class, objects, attributes with relationship obtained from importing the original Rational Rose model into a XML file and using XSL, the intermediate association expressing hierarchical relationship among model's objects, relationship, type, attribute for facilitating class bindings or XML messaging; hence the effect of using model relationship and objects interrelationship from the UML expressed in XML form is indicative that the XML has been parsed in order to yield the intermediate representation. Parsing of XML content expressed as schema object model (Fry: para 0030-0031) is disclosed in a similar approach by **Fry** use of JAXB data binding facility/algorithm to bind the schema into Java classes (Fry: para 0026, pg. 2; para 0045), where, in Fry, the XML is validated by a schema-aware parser to derive primitive or generic classes or into a DOM (para 0024, 0039) or a SOM (para 0041), the class binding derived from the XML/schema validating/parsing yielding intermediate primitive/generic classes that when instantiated would facilitate a framework such as to yield runtime API supporting certain primary operations (para 0025-0026), wherein, similar to the intermediate hierarchical representation as in Mestre, the SOM includes classes (extracted in a intermediate stage) enabling the DOM methodology to instantiate into APIs or java interfaces to allow other tools to operate (para 0043, para 0035); hence XML is disclosed as being parsed by a JAXB to enabling class bindings to support runtime code generation of an application, the binding yielding DOM instances of API to support other tools to operate, including marshaling/unmarshalling (see Fig. 1-2). The concept of primitive code associated with DOM is further disclosed in Worden.

Worden discloses the well-known practice of DOM structure representing a schema with associated APIs (Worden: para 0003-0005, pg. 1) where class-model based APIs calls into the XML to access objects therein (Worden: para 0034, pg. 3; *API which ... accesses or creates* - para 0045, pg. 4) to get information (Worden: para 0034, pg. 3) similar to reflection API mentioned in Mestre (see Mestre: para 0054); or to reflect XML meaning (Worden: para 0064, pg. 5; para 0175-0177 pg. 10) in which class or Java APIs are tightly associated with the DOM as in Fry and in return are available for the developers to create code to manipulate objects or semantics inside the XML schema.

As Java binding using XML schema and the associated DOM methodology were well-known at the time the invention was made, it would have been obvious for one of ordinary skill in the art to implement the information extracted from the XML representing the UML model in Mestre, so that this information deriving/extracting would use a parser as in Fry's class binding framework because the very development primitive and generic classes included within the XML parser in light of the DOM approach as by Fry and Worden, would enable primitive classes to be instantiated to help the runtime generators as in Mestre to yield instances of development tools APIs or other extended classes to access, modify, or develop the very data defined in the imported UML then converted model (as via XMI 32 in Mestre – Fig. 3) according to XML protocol/grammar, using the schema-aware parser approach in Fry according to well-known methodologies like JAXB, DOM to support the model class bindings and use thereof by the DOM (including underlying Java code to access and introspect class gathered from XML construct as in Worden), as set forth above.

Mestre does not explicitly disclose derived API code as *metadata API*, the metadata API *for enabling development tools to access the development objects to develop the application*.

The generator in Mestre's API is disclosed as providing code to marshall/unmarshall to serialize data as messages, as well as code to operate on databases, or extended to support a variety of applications including integrating modeling tools, type conversion, reuse of messages, optimization of code, retargeting optimizes database queries (see para 0105-0106, pg. 5; Fig. 1), the support of the amount of Java code instantiated amounts to an application interface environment to support further development of applications, including class bindings from the very XML input to yield XML messages or to serialize data back into XML form (via marshaling – see Mestre: para 0097; para 0103); hence the metadata aspect of the amount of code formed inside Mestre's API resulting from code generating based on the intermediate objects is recognized, since using XML schema to derive objects instantiated from classes compiled from the XML, and this metadata aspect is also evidenced in the creation of runtime objects to marshall SOM data back into metadata document is also disclosed in Fry's round trip approach (Fry: Fig. 1; para 0045). Further, use of a meta-implementation layer is disclosed in Severin.

That is, **Severin** discloses integration Engine with a meta layer interposed between the original model and the real implementation of classes/methods derived from the virtual implementation formed within the meta-implementation layer (Severin: para 0252-0257, pg. 22) that maps properties and descriptors being structured in the original model, the meta-implementation layer serving as basis for metamodel metadata to be captured (Severin: para 0109-011) as meta-form code (e.g. Severin: accessor, listener – para 0403-0404, 0341) to be identified – as via mapping of one-to-one relationships – the meta-implementation or accessor used to access the real source code to be compiled into real operations supporting database or legacy applications, service commands, security-related commands or resources access by way of XML conversion (Severin: para 0550-0562). The meta-implementation layer is further to disclosed as to map the relationships depicted the model similar to UML (Severin: para 0450-0452; Fig. 13) using descriptor's handler associated the model to identify a source format (Severin: para 0522-0525; serializer for each virtual implementation – par 0431) to help establish a metadata like connectivity view, based on which to compile runtime corresponding code (Fig. 5-34), the source format or metadata information forming the virtual implementation in order to be translated --by Severin's integration engine --into runtime real-implementations, including compiled code supporting operations extensible to various applications (Severin: object-XML mapping, database operations, relational mapping - para 0402-0408) including metadata-based marshalling/unmarshalling similar to that of Mestre (see Mestre: para 0107), the language conversion in Severin using serializer or database reconversion (Severin: para 0435-436) similar to marshal data into persisted XML or database; hence similar to the intermediate step in Mestre by which model relationship are captured into an ensemble to be inputted for code generators, Severin, like the metadata/database mapping and XML roundtrip reconversion in Fry, discloses an interface layer to derive correspondence of model objects association and to represent the derived metadata as a collection, with generation of handler or accessor objects which are used to find and to feed the actual code for implementing instances of executables supporting a variety of applications similar to that of Mestre or Fry.

It would have been obvious for one of ordinary skill in the art to implement the collection of code generated from the intermediate objects in Mestre's approach, such that based on the metadata nature of generated entities (Java code of Mestre's application interface) to extend the applications in different domains as set forth in Mestre and Fry, the ensemble of model-based and derived code would form a metadata layer of ensemble serving/layer as in Severin, as application program interface from which primitive classes (see DOM objects in Fry or DOM investigation APIs as in Worden) or accessor code (as in Severin) can be further extended or compiled into extended development tools to access or to manipulate data related to the initial model; i.e. metadata API enabling development of more tools to support or customize development of applications directed to various endeavors, as shown in Fry and Severin without having to make available pre-established source code, the tools using the metadata API code as accessor or extensible generic code for accessing development objects as shown in Fry and Severin.

As per claims 3-4, Mestre discloses (refer to claim 1) wherein the second language comprises Extensible Markup Language (XML); wherein the first language comprises unified modeling language (UML).

As per claim 6, Mestre does not explicitly disclose wherein the first language comprises a customizable extension. But based on the effect of generating API and the extension based thereon to change, delete data prior to marshalling the data back into a XML form as in Mestre and/or Fry, as well as the meta-data aspect by which accessor code in Severin can be compiled into operations supporting a variety of applications, including language or database conversion or type adaptation, it would have been obvious for one of ordinary skill in the art to implement the descriptor or association of object or object-oriented classes within the UML methodology in Mestre so that these provide a well-known internal OO language capabilities such as data encapsulation with one-to-many object instantiations, method override, interface extension, data serializability, abstract class implementation, all of which extracted as metadata serving as input into the metadata API environment, as set forth in claim 1, so as to enable additional object instantiations via extending the basic/generic classes in order to provide the customizable object instantiations responsive to the intended applications or required resource accessing related to data thereof as depicted in the rationale of claim 1, using the metadata capability of generated APIs shown Mestre, in view of Fry and Severin.

As per claim 7, Severin discloses accessing via reflection data regarding a package of code (Severin: reflection, constructor - para 0352, 0359) hence it would have been obvious for one of ordinary skill in the art to implement the extension set forth in claim 6, so that the customizable extension being part of the metadata API in Mestre would be able to implement an additional feature of the API via introspection on the intended code or retrieving constructor information for constructing a class or object, which would fall under the well-known methodology such as Java reflection API mentioned in Mestre (see para 0054).

As per claim 10, Mestre discloses a non-transitory_computer-readable storage device storing a computer program product, the computer program product being operable to cause a data processing apparatus to:

receive a metamodel in a first language (refer to claim 1), the metamodel describing a diagram of classes that define one or more development objects, the development objects representing building blocks (refer to claim 1) for developing the application, wherein the first language comprises a unified modeling language (refer to claim 4);

convert the metamodel to a model description that describes the metamodel in a second language according to an interchange format (refer to claim 1), wherein the second language comprises Extensible Markup Language (XML – para 0088; model.xml - Fig. 3).;

generate a set of intermediate objects to represent the classes of the metamodel (refer to claim 1); and

generate code using the set of intermediate objects as inputs to derive a Application Program Interface (API), wherein the API enables development tools to access the development objects to develop the application (refer to claim 1).

Mestre does not explicitly disclose generating a set of intermediate representing classes of the metamodel *by parsing the model description*. But this has been addressed in claim 1.

Nor does Mestre explicitly disclose API code as *metadata API*, the metadata API *for enabling development tools to access the development objects to develop the application*. But the metadata API -- for enabling tools to access development objects to develop the application – has been addressed in claim 1.

Nor does Mestre explicitly disclose metadata API *including an XML schema* that enables implementation of development of objects.

The code generated in Mestre's approach is disclosed as to follow a usage sequence including unmarshalling Java to XML and to marshal java to XML at the end of the cycle where data is serialized back as XML (XML to SQL, and back - para 0078), the persisting of XML or reuse model aligning with the methodology using XMI (Fig. 3), such that code for marshaling/serializing is adapted based on the association meta-information represented as hierarchy of relationship also implemented in a XML format (para 0091, 0093), the necessary presence of a XML type of representation leading to marshal code and serving as basis for Java code to serialize data back into a persisted form for reuse is recognized. Validation protocol for enforcing proper extensible markup syntax and its schema which underlies any use of XML representation of business or application models was well-known in W3c methodology, and this is shown in Fry's schema-aware parser and/or JAXB algorithm that provide inherent capabilities to validate or report discrepancies between a XML file and its schema(Fry: para 0023, para 0026), the JAXB supporting the round trip by which Java based on the dynamic binding, is reverted back to its XML form (Fry: para 0045), which is analogous to Mestre's sequence for reverting Java SQL back to its XML form. Based on the dual presence of a schema and XML file and data stream with respect to properly implementing XML for transmission or for storage, it would have been obvious for one of ordinary skill in the art to implement the metadata API marshal aspect so that marshal Java code would be supported by a *schema* having XML format, included with the API, in order to make use of the constraint and syntax validation that accompanies any attempt of unmarhalling XML stream or to marshal (serialize data back in XML form – e.g. the marshal functionaligy using the "metadata API" code – as set forth above - to effectuate the round trip mentioned in Fry-- for this XML form to be persisted for reuse or to be communicated along the distributed applications – e.g. retargeting of model, converting types between databases, authoring models - as contemplated in the multi-tier architecture, web-based, SOAP based multi-applications aspect of Mestre's framework (i.e. *XML schema* that enables implementation of development of objects within the scope of the metadata API) , where XML and messages underly compliancy with respect to any associated HTTP/W3c protocol.

As per claim 12, refer to claim 3.

As per claim 14, Mestre does not explicitly disclose wherein the set of intermediate objects comprises Java objects; but classes and objects represented as graph of relationship to provide meta-information for objects instantiation to implement marshaling/unmarshalling constitute the intermediate data or entities associated with the helper *functions* to enforce type conversions (see Mestre: *helper* functions - para 0090-0092). **Fry** discloses objects underlying a DOM methodology or JAXB to support validation (Fry: para 0026) or for manipulating the XML tree (Fry: para 0039), and whereas analogous to the helper code or reflection code in Mestre (para 0054), **Worden** also discloses reflection code associated with implementing a parse tree to help discover and get information (Worden: para 0034, pg. 3; *API which ... accesses or creates* - para 0045, pg. 4). It would have been obvious for one of ordinary skill in the art to implement the intermediate objects as taught in Mestre's helper and type conversion context, so that well-known XML/DOM tree associated Java objects are integrated along with the helper functionality, based on the Jax/DOM objects by Fry and reflection APIs by Worden; because these intermediate Java APIs support the parsing and tree building of a XML schema, as well as manipulating the tree nodes and validating the corresponding node of the XML-derived tree, and where the type related to class of the tree elements are introspected for proper construction via use of reflection APIs as mentioned in Mestre and in Worden, in a way that the intermediate objects coupled with these very APIs would constitute the elements needed to support the code generation leading to Mestre's metadata API capability to access data (e.g. getters, setters - para 0102) serialize database data, author models, reconvert data types and/or retarget model (para 0105) using the XMI approach.

As per claims 15-16, Mestre discloses (para 0091, 0093) wherein the XML schema includes a tree based on relationships in the metamodel; wherein the XML schema includes a reference based on an relationship in the metamodel first model (Classes, para 0080-0084); but does not explicitly disclose aggregation relationship or association relationships within the metamodel. The standard using Rational Rose modeling in Mestre (para 0032) entails UML constructs with relationship such as IS-PART-OF or IS-A, or one-to-many or one-to-one or many to one, as this syntax established by OMG was well-known at the time the invention was made; and even the model used in Severin is expressed as having aggregation and association (Severin: para 0037, 0039) type relationship. It would have been obvious for one of ordinary skill in the art to implement UML metamodel in Mestre so that the XML tree-representation thereof would refer one element of the model based on relationship such as to aggregation and association; because using the existing construct underlying the OMG methodology and UML modeling tool would obviate programming resources in Mestre's framework.

As per claim 17, Mestre does not explicitly disclose wherein the XML schema includes a complex type extension based on an inheritance relationship in the metamodel.

Based on Severin meta integration requiring mapping of complex association with need for new type creation (complex , new type - para 0086, 6; para 0186) among UML type hierarchies, it would have been obvious for one skill in the art at the time the invention was made to implement the XML schema intended to be reused in Mestre’s framework so that reference to association relationship to a UML and complex type extension are also represented in order to address the type extension and association needed within defined UML hierarchy, as by the mapping and integration (as in Severin) wherein integrating the schema would need to address complex processes requiring extension into new complex types, whereby to achieve instantiation of complex model associations – e.g. relationship among persisted records of a relational DB - that would befit Mestre’s endeavor to extend the use of the framework to a variety of model data format, database applications and multiple type conversions as well as XML messaging to support database reconfiguration.

As per claim 18, Mestre discloses a non-transitory_computer-readable storage device storing a computer program product being operable to cause a data processing apparatus to:

receive the metamodel describing a diagram of classes that define one or more development objects, the development objects representing building blocks for developing the application;

generate an Extensible Markup Language Metadata Interchange (XMI) model (refer to claim 1) that is a representation of the metamodel according to an interchange format (XMI – para 0080, pg. 4; para 0086 pg. 4);

generate a set of intermediate objects to represent the classes of the metamodel by parsing the XMI model using an Extensible Markup Language_(XML) parser; and

generate code using the set of intermediate objects as inputs to derive a metadata API enabling development tools to access the development objects to develop the application;

all of which having been addressed in claim 10.

As per claim 23, Mestre discloses wherein the metamodel is stored on **one of** a storage module (reuse … UML … in XMI – para 0059; exported from a UML – para 0060 – Note: model file being exported using XMI reads on storage module – see: *this model can be exported-* para 0078), a server, and a portable storage device.

As per claim 24, refer to claim 23

As per claim 25, Mestre discloses computer-readable storage device of claim 1, wherein the set of intermediate objects comprises Java objects (para 0090, pg. 4; helpers (functions - para 0092).

As per claim 26, refer to claim 25

4. Claims 8-9 are rejected under 35 U.S.C. 103(a) as being unpatentable over Mestre et al, USPubN: 2004/0015834 (herein Mestre) in view of Fry et al, USPubN: 2003/0163603 (herein

Fry), Worden, USPubN: 2003/0149934 (herein Worden) and Severin, USPubN: 2005/0005261 (herein Severin); further in view of Kadel, JR et al, USPubN: 2002/0184401(herein Kadel)

As per claims 8-9, Mestre does not explicitly disclose wherein the additional feature comprises an indication of a file border; wherein the API comprises a copy and paste operation.

Using XML mapping or representing model hierarchy as XML schema for converting languages between remote applications (Mestre: HTTP, SOAP, SQL, XML - para 0004, 0048, 0054) as in Mestre for supporting, in particular, database applications (Mestre: para 0101) is also further disclosed in Fry's web services (Fry: WSDL - para 0023) and Worden business processes (Worden: Fig. 7-9), where XML is the neutral representation expressing a business process or database transaction or retargeting applications involving different language format/type (Mestre para 0105). Kadel discloses a framework similar to message, HTML and XML interchange approach in Mestre's database applications, where XML schema provide relational-matching to correlate information between databases (Kadel: para 0083-0085), including type conversions (Kadel: Fig. 30); where attributes from a source model is exposed via an intermediary layer - XIS framework, mediation API - Fig 1-2– where metadata descriptive of application data is captured for use by the consumer module (para 0090-0093), where the metadata can be overridden (Kadel: para 0165) for a desired rendering purpose, and where reflection or accessor methods (Kadel: para 0172; Fig. 27B; Fig. 36) are used to dynamic support the declarative metadata support on the XIS mediation layer, similar to the concept of using DOM as in Fry or Worden, and wherein Kadel discloses embodiment of the framework as an authoring interface with drag-and-drop by the user as well as cut-and-paste capabilities (Kadel: para 0078), including editing of commands on GUI components, whereby the user can instantiate operation provided by the JAF API (see para 0349-0359, pg. 30; *canPaste()* Fig. 33b; *cut(clipboard)* Fig. 33c). Further, Kadel discloses API for JPanel package that operates on GUI component in terms of resizing, repainting, reshaping, paint Border, set Bounds, set Opaque (see awt.Component, JComponent, awtContainer, - Fig. 33E) hence the identification of Gui file border in order to manipulate its graphical rendering or browser representation, Windows GUI content so well known in interface operating with network or business transactions as set forth above is disclosed.

Based on the Microsoft context of Mestre's and Java classes underlying Mestre's computer system coupled with standard input devices, and the browser/network aspect in Mestre by which XML data is created to implement a visual model in relation to SOAP, HTTP, SQL (see Mestre: para 0074-0078; *update ... graphs* – para 0101), it would have been obvious for one of ordinary skill in the art to implement Mestre's java-based and internet applications in view of the extensibility of code for multiple applications mentioned in Mestre (e.g. authoring of models, retargeting models, graphs to describe messages – para 0105) so that (i) a feature included in the Mestre's metadata API would include a file border related to a rendering context as set forth from the above, because this would help identify the target file upon which *awt* operation or painting methods would be defined; and (ii) so that metadata and exposed Java classes libraries underlying java-based multi-tier, multi-applications framework of Mestre - in view of JAF API (Fig. 33) as in Kadel - are combined to support the creation of API type of operation to actually edit the attributes or manipulate exposed meta hierarchy using the standard GUI fabric (e.g. via *copy-paste* functions of GUI components), because this authoring capability falls under the intended use of Mestre's API, and would constitute efficient use of metadata and reusable Java packages whose utilization would be consistent with the extensibility aspect of the XIS framework, the extensive editing role played by user (see Mestre: authoring of these models – para 0105), and the availability of JAF API as set forth above.

5. Claims 20-22 are rejected under 35 U.S.C. 103(a) as being unpatentable over Mestre et al, USPubN: 2004/0015834 (herein Mestre) in view of Fry et al, USPubN: 2003/0163603 (herein Fry), or Worden, USPubN: 2003/0149934 (herein Worden) and Severin, USPubN: 2005/0005261 (herein Severin); further in view of Mullins et al, USPubN: 2003/0208505(herein Mullins) and Hejlsberg et al, USPN: 6,920,461 (hereinafter Hejlsberg)

As per claims 20-22, Mestre discloses database operations such as create, delete, search, update but does not explicitly disclose:

create a new development object as a transient object without an existing corresponding file; and modify the transient object until the transient object is committed to a persistent file;

instructions to destroy the transient object if a delete command is requested before the transient object is committed to a persistent file.

instructions to mark the persistent file as deleted if a delete command is requested after the transient object is committed to a persistent file.

Mullins discloses sourceforge.net tool (Mullins: para 0198) associated with database operations, including mapping one data source format into another with support of XMI (para 0104; para 0144-0165), the relational to relational data correlation with support of repository of UML, XML source exportable from and importable back into the repository via XMI (para 0101, 0104), the exporting in view of an user-editing context, including applying mapping rules against an associated schema (para 0108-0114) for compliance of the XML syntax, which is in line with schema validation and Java/XML conversion round trip in Fry (Fry: validating 228 – Fig. 2; para 0045) and the model binding and serialization back into XML form in Mestre's approach for optimizing database applications (Mestre: para 0106; 0079). **Mullins** also discloses database basic operations reminiscent of those implemented in Mestre's API (para 0100) including a *commit* operation (*Mullins:* para 0195, 0219, 0222) implemented via Java APIs (para 0238), the cache committed data subsequently transferred to a database via a "commit" operation (Mullins: para 0263).

Analogous to Mestre's implementing java code for database operations (e.g. create, delete, update) and use of .net methodologies as in Mullins, **Hejlsberg** also discloses a development application interface operating on layers or namespaces that expose class libraries or enumeration of related data structures or code constructs or tables (see Hejlsberg: col. 6; Fig. 2). Accordingly, Hejlsberg discloses application code instantiation from the libraries of reuse classes or OO packages (e.g. C++, Jscript, Microsoft ".NET" APIs) including UI objects with procedures to save a view, to customize drawing or drag-drop (col. 7, lines 48-62; col 8 lines 22-50) and a SQL namespace to interface with a database (col. 8 line 50 to col 9 line 11) including procedures to validate proper constructs, for implementing operations as to commit, dispose, rollback, save, accept/reject changes, cancel Edit (RejectChanges, acceptChanges – col 55; commit, delete, rollback – col. 57; CancelEdit col. 64; Delete, AcceptChanges – col. 65; Dispose, Finalize – col. 289; Commit, Dispose, Rollback, Save - col. 326).

Based on the use of Java APIs to support database application in Mestre as well as proper persisting of XML-structured model data back into a database in light of the teaching by Fry and the use of XMI as in Mestre for exporting persisted model file, it would have been obvious for one skill in the art at the time the invention was made to implement the code or APIs for supporting database operations in Mestre based on .net libraries as practiced in both Mulllins and Hejlsberg, such that Mestre's authoring of model data would include **a)** creating anew an element in a transient state being subjected to modification, and **b)** after user modifications, in accordance with the scheme of schema binding, or XML objects validating, information -- as shown in Fry or Mullins, or user-driven customization of model via GUI-event acts such as update graphs objects – see Mestre: para 0101 -- the final state of the transient element would be determined by tool capability or APIs with methods to *destroy* a transient object OR c) to *commit* the alteration to the model or the so-modified graph element (e.g. that are marked as to be retained) to a persistent form or database, i.e. the commit as taught in Hejlsberg and/or Mullins from above. One would be motivated to do this (i.e. create APIs by the user to destroy a transient object if it is not made for persistence committing) because that way the tool would provide more flexibility to the user to retain or undo a modification, where the support thereof in form of created APIs would enable changes made in Mestre's graph modifying and model retargeting approach to be reconsidered for user satisfaction prior to being committed, and/or allowing removal of undesired implementation gathering of data, whereby obviate potential runtime errors should actual translation of uncorrected constructs become finalized, because only properly formulated/accepted data would be kept in a database for a reuse purpose; e.g. using XMI and model export (see Mestre: para 0078), as this would fall into Mestre's concern of optimizing database SQL messaging or data transfer (para 0106)

Response to Arguments

6. Applicant's arguments filed 10/03/2011 have been fully considered but they are mostly **moot in light of the new grounds of rejection** which have been necessitated by the Amendments, following are remarks that would address some more particular arguments.

(A) Applicants have submitted that the API in Severin are existing APIs readapted to match the descriptor of a metamodel, not APIs that are derived from a set of intermediate objects (Applicant's Remarks pg. 13-14), and that Worden’s mapping does not teach deriving of metadata API as claimed. The use of Severin is now implemented to address the teaching that the API generated based on the claimed “deriving” amounts to a mere collection/environment acting as container of derived Java APIs or marshalling code or XML schema. The Applicants retort to the previous claim language objection regarding misuse of "deriving" (Applicant's Remarks pg. 9: *generated code is included*) has it made clear that the metadata API is actually a container or a collection of elements, and this is evidenced by the Disclosure in element 130 of Figure 11, where the actual "deriving" is implemented by the generators 1135, 1140, 1145 whose outputs are collected as interfaces 130a, marshaling code 130b, and schemas 130c. The 'deriving' has been interpreted as mere generator of code that yield interfaces, Java API or marshalling code etc. all of which gathered in the layer referred to as metadata 130, which is not an “Application Program Interface” per se, but a collection of a plurality thereof. Nowhere in the Disclosure teach how API 130 operates as a genuine API, i.e. the contents of this API 130 are used as interface to invoke further development actions, and this understandable. The argument that no references teaches an actual generation of one single API referred to as “metadata API" (see the collection nature of this API 130 in Fig. 11 of the disclosure) would be deemed largely non-convincing. Severin actually teaches an entire layer that accrues a number of APIs to support further java code used for development; i.e. model object-related metadata accessing functionalities within the course of the development. Besides, Severin is not a main reference used to anticipate the newly feature recited as "Metadata API"; that is, the grounds of rejection has been modified in view of the Amendments (emphasis added). Interpretation of the added feature has been based on the Specifications, and deriving a single API 130 based on the intermediate objects is deemed not supported (see Specifications: middle pg. 30). The Metadata API has been construed as a layer that gathers the results from executing *generators 1135, 1140, 1145*; i.e. this metadata API <u>not actually generated</u> (emphasis added) by said generators, said metadata API never disclosed <u>as operating as a genuine API</u> (emphasis added) according to the Disclosure or one of ordinary skill in the art grasp of this "Application Program Interface" concept. Notwithstanding that, Applicant's arguments fail to comply with 37 CFR 1.111(b) because they amount to a general allegation that the claims define a patentable invention without specifically pointing out how the language of the claims patentably distinguishes them from the reference.

(B) Applicants have submitted that Hejlsberg fails to cure the deficiencies of Kadel, Severin and Worden, because of the "deriving" feature not fulfilled by those references (Applicant's Remarks pg. 14-15). This also has to be considered moot for the same reasons set forth above.

Conclusion

7. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Tuan A Vu whose telephone number is (571) 272-3735. The examiner can normally be reached on 8AM-4:30PM/Mon-Fri.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Lewis Bullock can be reached on (571)272-3759.

The fax phone number for the organization where this application or proceeding is assigned is (571) 273-3735 (for non-official correspondence - please consult Examiner before using) or 571-273-8300 (for official correspondence) or redirected to customer service at 571-272-3609.

Any inquiry of a general nature or relating to the status of this application should be directed to the TC 2100 Group receptionist: 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Tuan A Vu/

Primary Examiner, Art Unit 2193

December 10, 2011